Extended Essay in Computer Science

The Use of Machine Learning to Classify Music Genres

To what extent can a fine-tuned DistilHuBERT transformer model accurately classify music genres using the GTZAN dataset compared to a Support Vector Machine (SVM)?

Word Count: 3991

Session: May 2025

CS EE World
https://cseeworld.wixsite.com/home
May 2025
28/34
A
Submitter Info:
Anonymous

Table of Contents

1 Introduction *3*

2 Background Information 4

2.1 Music Genre Classification 4

2.1.1 Feature Extraction 5

2.1.2 Model Training 5

2.1.3 Classification 6

2.2 Support Vector Machine 6

2.3. Transformers 9

2.3.1. Transformers for audio classification tasks 10

2.3.2 HuBERT 12

2.3.3. DistilHubert 12

2.4 Data Set Used 13

3 Methodology *14*

4 Experimental Results *15*

4.1 Support Vector Machine Results 15

4.2 DistilHuBERT Results 17

5 Data Analysis *20*

5.1 Analysis on SVM's performance 20

5.2 Analysis on DistilHuBert's performance 22

5.3 Insights from both perspectives 22

6 Limitations *23*

7 Conclusion *24*

9 Appendix *26*

1 Introduction

Music is an art combining various instrumental sounds and vocals to produce harmonies expressing many emotions plays an important role in almost everyone in this world resulting to a wide range of music styles. Individuals are drawn towards these different musical genres that resonate with their unique tastes. To accommodate these diverse interests, music apps and other media platforms today use different algorithms to classify and recommend musical genres. Classifying music into different genres is one of the branches of Music Information Retrieval (MIR). It's a complex and time-taking task which involves steps like feature extraction, model training, embedding, classification of the genres, (GeeksforGeeks) etc.

Existing machine learning models, which are algorithms that train on datasets to recognize patterns and predict various outcomes, like Support Vector Machines (SVM) and Convolutional Neural Network (CNN) are well known for their performance in audio classification tasks (*GTZAN*). Recent advancements on transformer models, which are deep learning architectures designed for processing sequential data, have shown exceptional capabilities in Natural Language Processing, Computer Vision and audio tasks. A transformer model when fed with enough data, can recognize, categorize and predict intricate patterns. Such model used for audio classification tasks is the DistilHuBERT which is a smaller version of the HuBERT model, with a increased speed of 73%. (*Fine-tuning*). While CNNs are famous for their ability to classify audio through spectrograms which are visual representations of audio in waveform they need higher computational resources and larger datasets that are hard to perform. Even though models like CNNs, Recurrent Neural Network (RNNs), or other ensemble models are a go-to model for audio

classification tasks, to avoid any errors, I found SVM to be a better model to use for my paper because of factors like computational efficiencies, ease of use and accuracies. Therefore, this research paper seeks to compare the accuracy between the new, and faster transformer model with the old and frequently used Support Vector Machine algorithm. The essay plans to provide an overview of both the DistilHuBERT and SVMs, describing their experimental setup, training and evaluation and then putting them up against each other using the GTZAN dataset, comprising of 1000 audio clips spanning 10 different genres to see which one is the better one in music genre classification and provide valuable insight and potential implications for Music Information Retrieval. While researching, GTZAN was seen to be the most commonly used dataset for genre classifications, I decided to use the same dataset for my paper. However, inherent bias in the dataset, and lack of diversity was acknowledged as factors that could affect the results for fairness. Which brings me to my research question, "*To what extent can a fine-tuned DistilHuBERT transformer model accurately classify music genres using the GTZAN dataset compared to a Support Vector Machine (SVM)?*". The question is worthy of investigation because it challenges the existing theories and viewpoints on audio classification tasks and sheds light on more efficient methods that could enhance productivity and resource management.

2 Background Information

2.1 Music Genre Classification

Audio classification tasks are one of the most common tasks in machine learning. They make use of different methods and algorithms for the most efficient result. The audio is represented with an audio signal which has the properties like – frequency,

decibel and bandwidth. But each song has its own signature like its rhythm, acoustics, danceability, tempo which makes that particular song unique. These differentiations can be categorized into something called genres. Moreover, these genres are classified because of certain challenges, such as searching for similar songs, developing playlists for song reference and identifying the type of race that enjoy listening to that music. Automated music genre classification takes care of all these problems, furthermore, reducing the load on humans to manually do this task. This process involves several steps ---

2.1.1 Feature Extraction

This step is to extract important features from a raw audio signal. This process normally involves the raw audio being processed and broken into different segments (*Andradaolteanu*). Then this preprocessed audio is passed through machine learning models to extract features like their frequency, amplitude, tempo and Mel-frequency cepstral coefficients (MFCCs).

2.1.2 Model Training

These extracted features are then used to train a machine learning model, such as a SVM or a CNN, to learn the different interpretations between the features and the corresponding music genre labels. These models are then trained on a labeled dataset, like the GTZAN dataset, where each music recording is associated with a ground-truth genre label. The choice of model often depends on the available computational resources, desired model of the user, flexibility, and complexity of the dataset.

2.1.3 Classification

The trained model classifies new audio files by predicting the genre based on learned features, assigning them to a specific genre category with a probability score. The model takes the extracted features of the audio file as a input and outputs a the predicted genre name based on the learned patterns in the training data. By analyzing the extracted audio features of the input audio, the model produces a predicted genre label that aligns with the patterns learned from the training dataset.

2.2 Support Vector Machine

Support Vector Machines (SVMs) are a cluster of supervised learning machines that play an important role to address classification problems. The figure below shows the mechanism of SVM, where the data points are separated to find the optimal hyperplane by maximizing the margin.

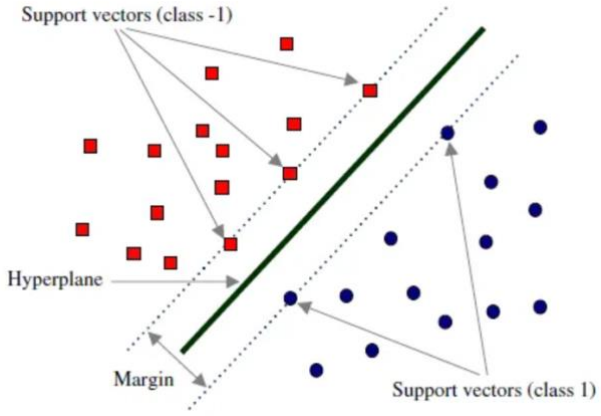

Figure 1: Support vector machine diagram

Hyperplane is a virtual space where data points are divided into two categories. The line that divides these points are called n-dimensional lines. In 2D, a straight line while in 3D it's a plane. But in cases like where the dataset is very complex the data

points cannot be linearly separable, in those instances one extra dimension is added into the n-dimensional plane. This hyperplane is just that, it maximizes the margin between different classes for better efficiency. In other words, this hyperplane decides the maximum distance between the two classes (the margin). The points closest to the hyperplane are called support vectors. (*Visually Explained*)

Hyperplane can be formed into an equation –

$$w^T x + b = 0$$

Where:

- w^T is the weight vector (normal to the hyperplane),
- b is the bias term (or offset),
- x is the feature vector (e.g., audio features like MFCCs).

The sign of this equation determines the classification:

- If $w^T x + b > 0$, classify as one genre (e.g., "Rock").
- If $w^T x + b < 0$, classify as another genre (e.g., "Jazz").

But there is no need for mathematically implementing this in to the code as libraries like Scikit-learn automatically implement these process' for you with help of predefined of functions. The function *svm.SVC()* takes care of fitting the data points into the model. The distance between the dotted lines and the optimal hyperplane is known as the Margin where the margin is the area which does not contain any data points. The margin is inversely proportional to ||w||. The main problem of SVMs is to minimize ||w|| to maximize the margin.

This leads to the mathematical equation (*Visually Explained*):

$$\frac{1}{2}||w||^2$$

But in this problem, few constraints can be added to confirm that the data points are on the right side of the hyperplane –

$$w^T x + b \geq 1, \qquad \forall_x \in C_1$$

$$w^T x + b \leq -1, \qquad \forall_x \in C_2$$

where, C_1 and C_2 refers to the two different classes which consists the data points for those greater than one and the other lesser than one. (*Visually Explained*)

But as most music genre classification data sets turn out to be non-linear, the normal linear optimization problem might not do the trick. We use something called as the Kernel Trick. In these cases, Nonlinear transformations are applied to data points before using SVM to achieve non-linear decision boundaries, overcoming SVM's limitation of flat boundaries. Then these data points will be linearly separate in a higher dimensional feature space. Then the linearly separate classes are arranged in the higher dimensional feature space, which are then mapped into a higher dimensional space without actually calculating, this is called the Kernel Trick (*Visually Explained).* Some of the common Kernel functions are – linear, polynomial and the Radial Basis Function (RBF). This trick is also taken care of the Scikit-learn library where by just mentioning the type of kernel and the required parameters it abstracts the entire process. In the code this can be implemented (*Andradaolteanu*) when training the SVM model like

```
svm = SVC (kernel='rbf', C=10, gamma=0.01)
svm.fit(X_train, y_train)
```

2.3. Transformers

Transformers are a machine learning models that have been revolutionizing major fields in AI like natural language processing (NLP), computer vision, and recently audio classification tasks. It was first introduced in the paper "Attention is All You Need" by Vaswani et al. (2017), where the transformer architecture was based on the self-attention mechanism, which is nothing but the ability of a model to compare consecutive data and make decisions accordingly.

Transformers consist of an encoder and a decoder. (*Unit 3*) The data input if first processed by the encoder where it takes the input to build a representation of features of that data set, to understands it. The decoder is where it makes use of both the features and data to generate desired outputs.

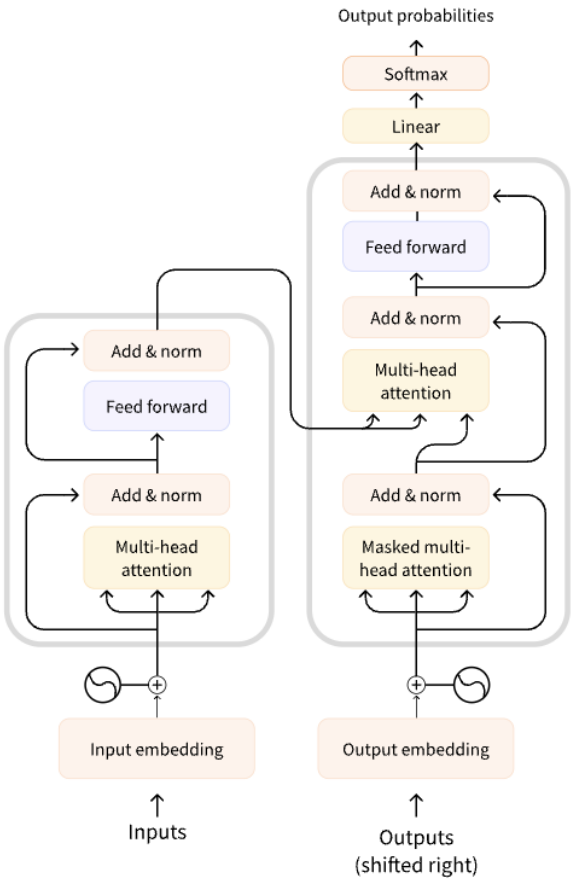

Figure 2: Architecture of a transformer

But there can exist encoder-only models, like the BERT model, which is good for classification tasks that require understanding. Transformers are built with something called attention layers, the “self-attention mechanism”. These layers act as an informant to the model telling it to work on certain elements only ignoring the rest in the input sequence, reducing workload to represent feature representations.

2.3.1. Transformers for audio classification tasks

In the context of audio classification, transformers have gained significant attention for their ability to handle long-range dependencies and capture intricate features in audio signals.

In these audio classifications audio is taken as the output and a class probability as given as the output.

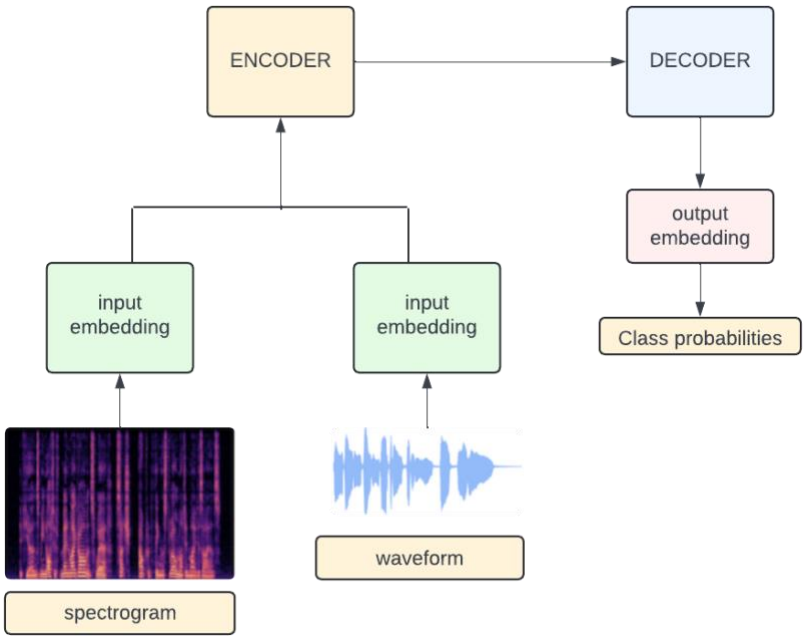

Figure 3: Transformers process for audio classification

These transformers take audio input in a waveform or in spectrogram form. A waveform is a visual representation of sound that plots sample values over time, showing how the sound's amplitude changes. (*Unit 3*) While, a spectrogram plots the frequency over time allowing one to see the time, frequency, and amplitude all in one graph.

Models like Whisper transform audio waveforms into log-mel spectrograms (The mel scale is a perceptual scale that approximates the frequencies audible by the human ear.), classifying them into 30-second sections. Each log-mel spectrogram is a visual 2D representation of frequency over the rate of change in time. This conversion shortens the sequence length (time frame), compared to the raw waveform. Consequently, spectrograms also reduce the input data size being more efficient than the raw waveform as it’s compressed.

The decoder in the transformer architecture outputs certain output embeddings. These output embeddings are then converted into class probabilities which is the objective of the transformer model (*IBM Technology*). This is achieved by incorporating a classification head, which generates probabilities across predefined classes relevant to the classification task.

Transformers excel in audio classification through their self-attention mechanism, capturing dependencies across sequences, useful for genre classification and speaker recognition. Unlike SVMs, they process sequences in parallel, improving efficiency. Pretrained models like HuBERT and Wav2Vec, (*Fine-tuning*) built on vast audio datasets, enhance performance and fine-tuning, making transformers highly effective for audio-related tasks.

2.3.2 HuBERT

HuBERT is a transformer-based model designed for supervised learning in audio tasks. It was introduced by Facebook AI Research (FAIR) to learn general-purpose speech representations from unlabeled audio data, making it highly effective for tasks like speech recognition, speech synthesis, and audio classification. HuBERT (Hidden-Unit BERT) is a self-supervised learning model that learns from unlabeled audio data by predicting "pseudo-labels" generated through clustering input features. (*Hubert*). This process enables HuBERT to extract meaningful audio representations without relying on labeled data.

HuBERT's training involves two stages: first, audio features are clustered into discrete units, and in the second, these clusters are used as targets for masked language modeling (similar to BERT). The architecture includes a multi-layer transformer encoder that extracts hierarchical audio representations, making HuBERT effective for downstream tasks such as speech recognition and emotion detection. One key advantage of HuBERT is its ability to train effectively with limited labeled data, relying instead on abundant unlabelled audio (*Hubert*). This makes it particularly valuable for applications where labeled data is scarce but large amounts of raw audio are available, surpassing traditional models in flexibility and performance across various tasks.

2.3.3. DistilHubert

DistilHuBERT, is a distilled or like a compressed version of the HuBERT model. This model is 75% smaller than the parent model and 73% faster (*DistilBERT*) in performing it's tasks. Moreover, DistilHuBERT requires little training time and data, opening the possibilities for efficient pre trained models and no waste in time.

2.4 Data Set Used

The GTZAN Genre Classification Dataset is a widely-used public dataset in the field of music genre recognition (MGR). It consists of a diverse collection of audio files and related visual and numerical data, specifically designed to help researchers understand and classify different genres of music.

The dataset includes 1,000 audio files, each file representing different music genres, with 100 tracks per genre. Each track is 30 seconds long and was collected from a variety of sources between 2000-2001, resources ranging from personal CDs, microphone recording to radio providing a versatile dataset in terms of recording conditions. (*GTZAN*) These genres include rock, pop, hip hop/rap, electronic/dance, jazz, classical, country, R&B/soul, reggae, and folk.

Alongside the audio files, the dataset provides Mel Spectrogram images, which are visual representations of the audio signals. (*GTZAN*) These spectrograms make it possible to apply Support Vector Machines (SVMs), which can work with numerical features extracted from the visual representations, to classify music genres based on these spectrogram-derived features.

Additionally, there are two CSV files in the dataset, each containing extracted audio features. One CSV provides features like the mean and variance for the 30-second audio clips, while the second CSV breaks the audio into 3-second segments, increasing the data size tenfold, which is advantageous for training machine learning models. (*GTZAN*)

Through exploratory data analysis (EDA), the dataset allows for the visualization of different genre characteristics in the form of waveforms and spectrograms, aiding in the understanding of what makes one genre distinct from another.

In summary, the GTZAN dataset provides a rich foundation for researchers aiming to analyze, visualize, and classify music genres using machine learning and neural network techniques.

3 Methodology

The GTZAN dataset was used, containing 1,000 audio files across 10 music genres. The audio files were transformed into Mel Spectrograms, and features such as mean and variance were extracted. Two models were applied: a fine-tuned DistilHuBERT transformer and a Support Vector Machine (SVM). The dataset is split into training (70%) and testing (30%) sets (Unit 3). A grid search was used to optimize hyperparameters using the RBF kernel because of its ability to capture complex non-linear relationships that are evident in the dataset. For SVM, audio features such as MFCCs were extracted before testing and a fine-tuned DistilHuBERT was imported from the Kaggle library which was fine-tuned directly on the raw waveforms of the same dataset. The accuracy of both models was evaluated and compared to determine which model performed better on classifying music genres. Python libraries such as Scikit-learn were used to visualize the data and predictions through confusion matrix, and class-wise prediction bar and pie graphs.

4 Experimental Results

4.1 Support Vector Machine Results

Output for SVM

```
Finished feature extraction for 999 files.
Converting features and labels to NumPy arrays...
Features shape: (999, 173), Labels shape: (999,)
Encoding genre labels...
Encoded labels: ['blues' 'classical' 'country' 'disco' 'hip-hop' 'jazz' 'metal' 'pop'
 'reggae' 'rock']
Scaling features...
Splitting data into training and testing sets...
Training set size: 699, Test set size: 300
Training SVM model...
Predicting on the test set...
SVM Accuracy: 63.67%
```

Figure 4: Snap of the terminal output of SVM

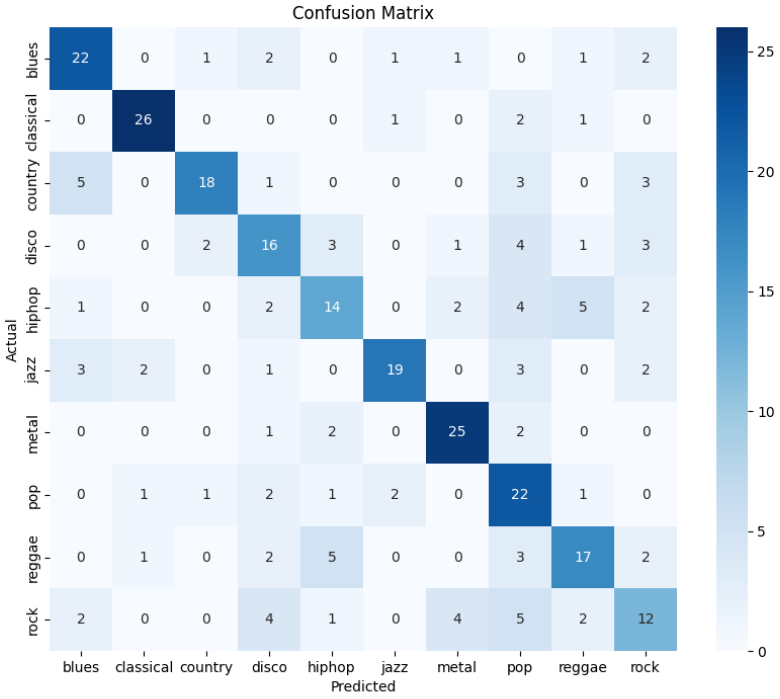

Figure 5: Confusion matrix of SVM

The SVM took a long time to process the data in which the feature extraction would stop due to some dataset dependencies, which required significant troubleshooting. The SVM showed an accuracy of **63.67** %. To visualize the performance of the SVM a confusion matrix was generated highlighting a concentration of rate predictions along the diagonal and class-wise prediction bar and pie charts to detect bias and trends in the model.

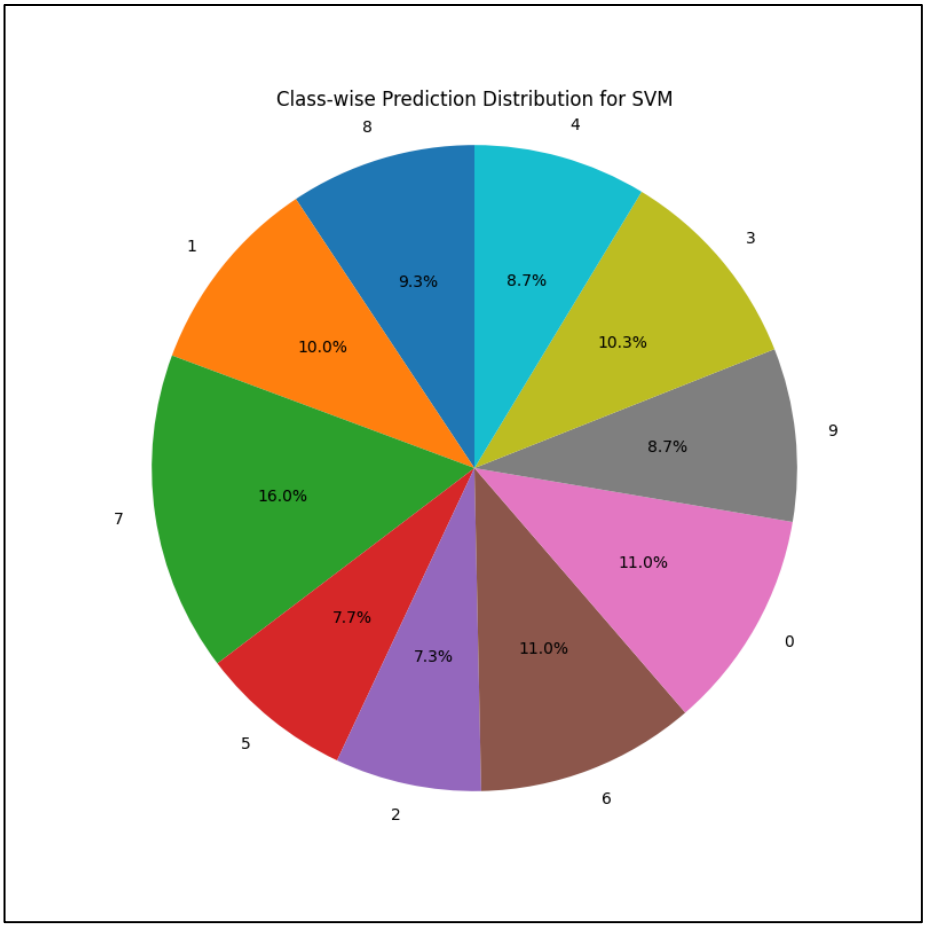

Figure 6: Class-wise Prediction Distribution (Pie chart) for SVM

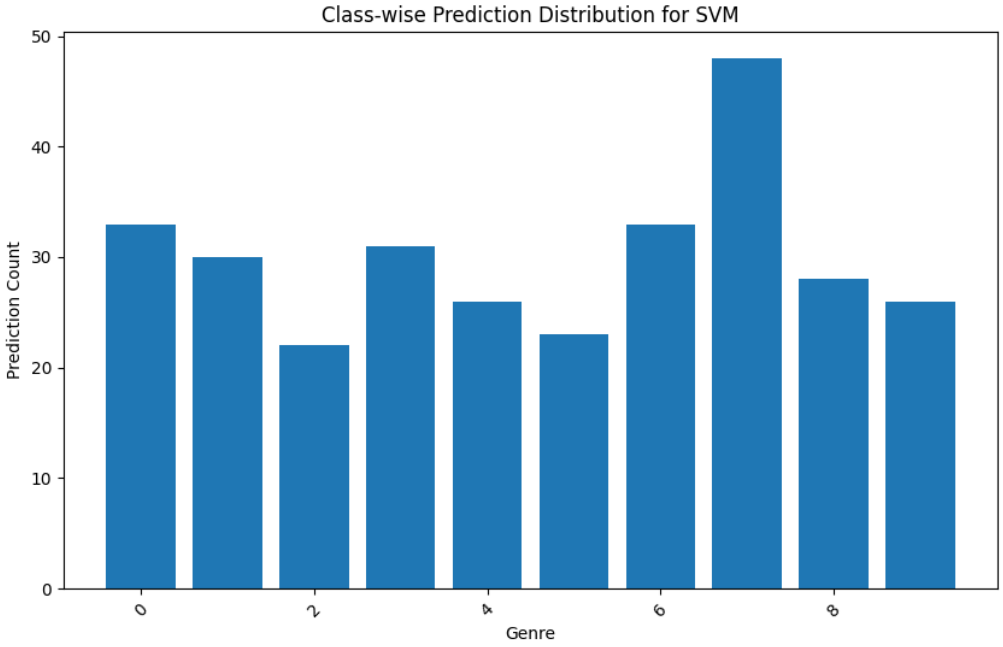

Figure 7: Class wise Prediction Distribution (Bar chart) for SVM

4.2 DistilHuBERT Results

Output for DistilHuBERT

Finished processing genre jazz in 42.71 seconds.

……

Finished evaluation on GTZAN dataset in 427.78 seconds.

DistilHuBERT Accuracy on GTZAN dataset: 94.99%

Generating confusion matrix...

Figure 8: Snap of the terminal output of DistilHuBERT

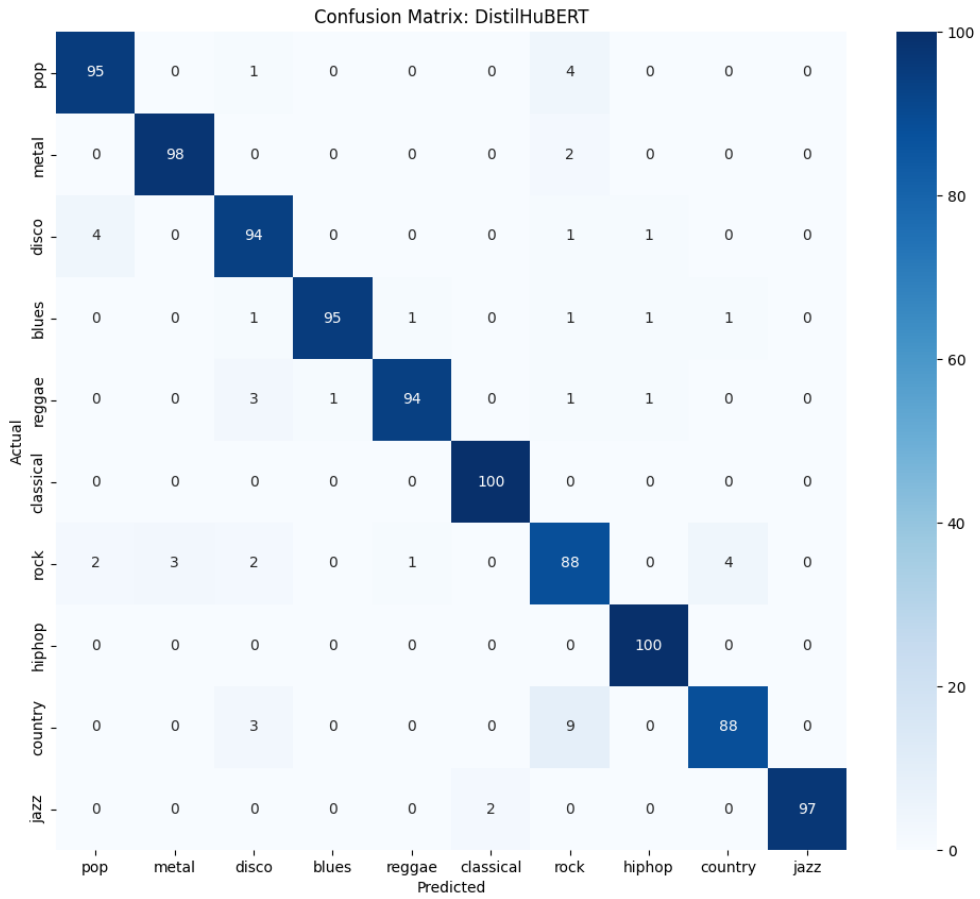
Confusion Matrix: DistilHuBERT
pop 95 0 1 0 0 0 4 0 0 0
metal 0 98 0 0 0 0 2 0 0 0
disco 4 0 94 0 0 0 1 1 0 0
blues 0 0 1 95 1 0 1 1 1 0
reggae 0 0 3 1 94 0 1 1 0 0
classical 0 0 0 0 0 100 0 0 0 0
rock 2 3 2 0 1 0 88 0 4 0
hiphop 0 0 0 0 0 0 0 100 0 0
country 0 0 3 0 0 0 9 0 88 0
jazz 0 0 0 0 0 2 0 0 0 97
Actual
pop metal disco blues reggae classical rock hiphop country jazz
Predicted
100 80 60 40 20 0

Figure 9: Confusion matrix of DistilHuBERT

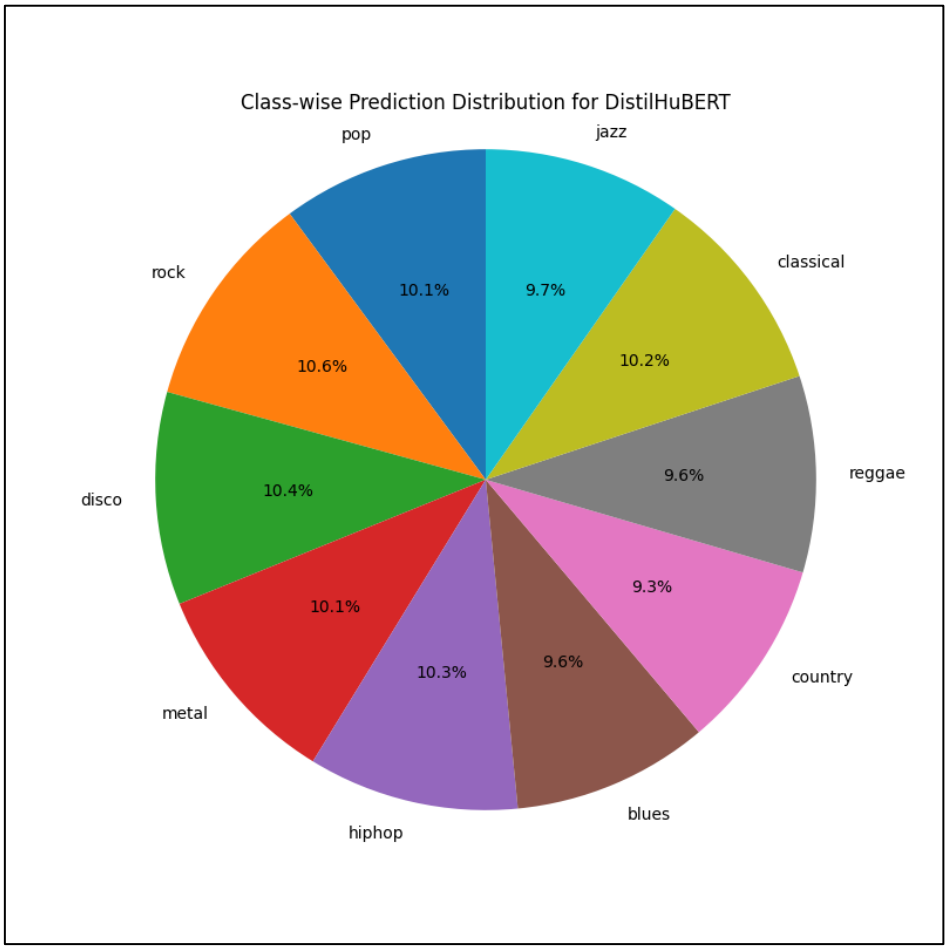

Figure 10: Class-wise Prediction Distribution (Pie chart) for DistilHuBERT

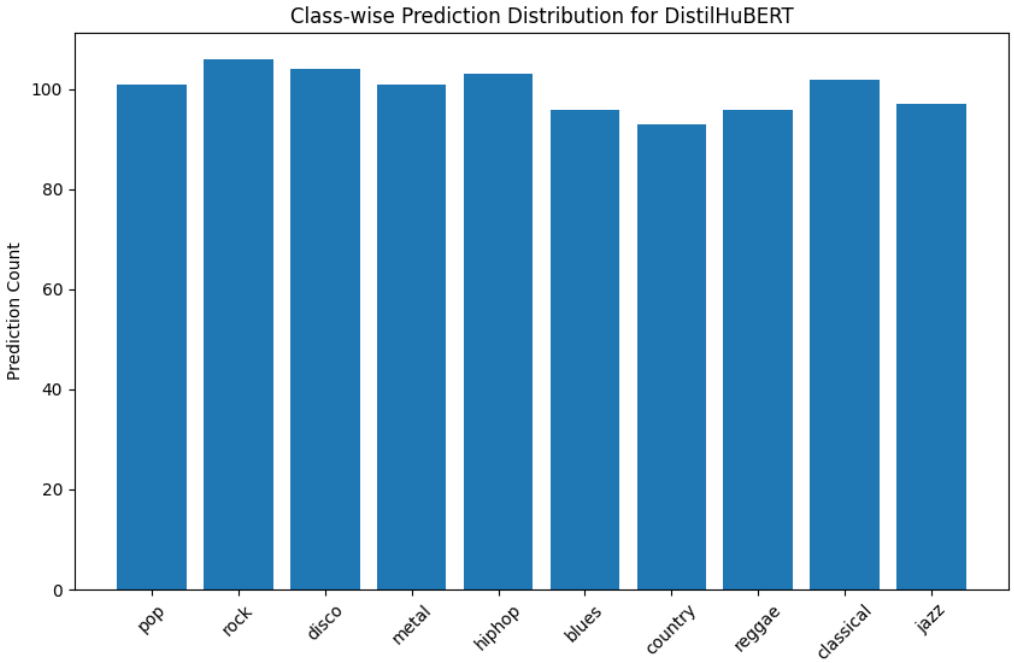

Figure 11: Class-wise Prediction Distribution (Bar chart) for DistilHuBERT

While experimenting for DistilHuBERT there were certain obstacles that may have affected the output like while installing the DistilHuBERT dependencies, I had trouble with compatibility issues with the FFmpeg and PyCharm discrepancies which required me to troubleshoot. But DistilHuBERT still achieved an accuracy of **94.99**%, outperforming the SVM. The confusion matrix and the output of DistilHuBERT revealed that the model accurately predicted 285 genres out of 300 test samples, with minimal confusion between the genres. It also completed the testing in 427.78 seconds faster than the SVM.

Model	Accuracy	Running time	Observations
SVM	63.67%	2280 seconds	Struggled with similar genres
DistilHuBERT	94.99%	420.75 seconds	Predicted accurately with slight errors between similar genres

Figure 12: Table comparing results of the two models

---

5 Data Analysis

5.1 Analysis on SVM's performance

While experimenting SVM manually extracted features which involved trimming silence and converting audio signals into numerical representations that could be fed into SVM. Out of 300 test samples, the SVM model made correct predictions for 191 files, with notable misclassifications in similar genres like pop, rock, and disco which

is evident in the confusion matrix. The SVMs low accuracy of **63.67**% can be because of its reliance on manually extracted features. It used features such as MFCCs, central spectroids that do capture important features of audio signals but they fail to recognize the patterns in the music scene.

From fig 4, we can deduce that out of 100 classical tracks only 80 were correctly classified, with wrong classifications across jazz, reggae and disco. Genres like metal and classical were the most accurate predictions with values of 26 and 25. But at the same time genres like disco was often misclassified as pop and hip-hop. Also, SVM does not adapt to data beyond predefined features, limiting its ability to generalize, for example SVM struggled to classify reggae and country due to their rhythmic and harmonic similarities. We can say that the model struggled to classify between similar sounding genres.

Looking at the class wise-prediction pie and bar charts (fig 5,6) I can see the tendencies in predictions, class representations and dataset bias. Genres such as pop (class 7) is the most predicted genre with 16% while country (class 2) was the least predicted genre taking space up to 7.3%. These imbalances suggest that the model is biased towards these genres, which may be due to reasons like how the SVM extracted the features, the percentage splitting of the training and testing dataset, or the dataset having any corrupted file. This suggests that SVM is overfitted towards pop because of its tendency to overpredict certain genres like pop and country. While classical was the most accurate predicted genre it was also the, it has similar musical characteristics and features with the overpredicted genre country, giving model more experience or overfitted to country-like genres. These challenges could be addressed if proper dataset precautions and the way the model was trained could be optimized.

5.2 Analysis on DistilHuBert's performance

In contrast, DistilHuBERT classified genres with an accuracy score of 94.99%, highlighting the advantages of using deep learning models. The model was pre-trained on the dataset, and unlike SVM it processed raw waveforms and learnt hierarchical representations of data, enabling it to differentiate between slight nuances like timbre, harmony, rhythm, etc.

From fig 7, the model classified classical, jazz and hip-hop with 100% accuracy, reflecting that the model can differentiate accurately between distinct audio characteristics but at the same time there were certain overlaps where the model mistook country as reggae and rock as pop or metal. Hence, the model performed well on genres that have distinct features while misclassified genres that have similar characteristics. This was also seen in SVM but I think due to DistilHuBERT's being pre-trained on the dataset, processing raw waveforms it was better at predicting genres accurately.

Looking at the class-wise prediction pie and bar charts (fig 8,9), we can deduce that the predicted genres are evenly distributed with every genre accounting to at least 10% of the predictions. As the dataset is uniformly distributed for this model, we can say that it does not have significant bias. The model has no significant bias towards the dataset, and performed with a high accuracy rate proving the power of deep learning transformers to distinguish between characteristics like rhythmic and harmonic differences.

5.3 Insights from both perspectives

DistilHuBERT achieved an accuracy of 94.99% outperforming the SVM 63.67% by a difference of 31.32% showcasing its superior ability to capture intricate patterns for music genre classification on the GTZAN dataset. Both models have a foundation in

Mel Spectrograms, but SVMs manual feature selection again caused a limitation on its generalization capabilities, whereas DistilHuBERT accurately extracted hierarchical features, enhancing classification accuracy. This adaptive feature extraction capability of DistilHuBERT was crucial in achieving its near perfect accuracy, but ethical considerations arise from its dependency on pretraining, which might've lead to embed bias on the dataset, potentially leading to over-fitting on dataset-specific patterns. This also downsides the underrepresented genres and cultural diversity in music classification raising concern about generalization to unseen data.

6 Limitations

The first limitation of this experiment can be the process of training between SVM and DistilHuBert. While the SVM was trained on the data set during the running of the experiment, DistilHuBERT was a pre-trained model on the dataset. Second, is the limitation of the dataset, it contains only 10 genres while in real life there are hundred evolving and yet to come. Third, is the SVMs and the transformer model limitations – SVM is limited to model non-linear relationships in high dimensional data and performance is highly relied on feature engineering, DistilHuBERT's dependence on the pretrained model, which may have introduced biases inherent in the original training data. Fourth, is the compatibility issues of the SVM and the DistilHuBERT models with the device that I used for running the code. If the device could incorporate the dependencies and run efficiently with large datasets there could've been better results. Resulting, in underscoring of the runtime in feature extraction in SVM, and predictions. Fifth, is the models' bias against the data, where in SVM it can be seen through the class-wise predictions that it predicted few genres more than others affecting the results' generalizability.

7 Conclusion

In this paper, the accuracy of classifying genres between SVMs and a transformer model DistilHuBERT was analyzed. Background information was provided for understanding of the models before experimenting which resulted to the DistilHuBERT outperforming the SVM model in terms of accuracy – 63.67% vs 94.99. However, these abilities come with notable trade-offs, including time constraints and computational costs. From a broader perspective, these results emphasize the critical role feature representation and use of dataset in machine learning tasks. I underestimated the importance of feature extraction and assumed SVM would perform competitively as it was a commonly used model in this research area but the stark performance gap highlighted how modern transformers excel in tasks requiring understanding of nuanced data structures. Therefore, the results answer the research question by demonstrating that DistilHuBERT is more accurate than a traditional SVM in music genre classification using the GTZAN dataset. While SVM remains viable for tasks with limited data and resources, the use of transformer model sets a new benchmark for accuracy in music genre classification. Future research can test larger datasets with dynamic and up to date music genres, maybe combine traditional features of the model with the deep learning for more efficiency and optimize fine-tuning with varied pre-trained data sets. Additionally, using hybrid feature extraction methods and transfer learning between models can help optimize and boost performance in MIR applications.

8 References

GeeksforGeeks. “Music Genre Classification Using Transformers.” GeeksforGeeks, 17 Nov. 2023, www.geeksforgeeks.org/music-genre-classification-using-transformers.

Fine-tuning. Fine-tuning a Model for Music Classification - Hugging Face Audio Course. huggingface.co/learn/audio-course/en/chapter4/fine-tuning.

Visually Explained. “Support Vector Machine (SVM) in 2 Minutes.” YouTube, 9 Sept. 2021, www.youtube.com/watch?v=_YPScrckx28.

“The Kernel Trick in Support Vector Machine (SVM).” YouTube, 9 May 2022, www.youtube.com/watch?v=Q7vT0--5VII.

Unit 3. Transformer Architectures for Audio - Hugging Face Audio Course. huggingface.co/learn/audio-course/en/chapter3/introduction.

IBM Technology. “What Are Transformers (Machine Learning Model)?” YouTube, 11 Mar. 2022, www.youtube.com/watch?v=ZXiruGOCn9s.

Hubert. huggingface.co/docs/transformers/v4.47.0/en/model_doc/hubert#overview.

Andradaolteanu. “Work W/ Audio Data: Visualise, Classify, Recommend.” Kaggle, 25 Mar. 2020, www.kaggle.com/code/andradaolteanu/work-w-audio-data-visualise-classify-recommend/notebook.

Pattanaik, None Raj Kumar, and None Answeta Jaiswal. “Classification of Music Genre Using Support Vector Machine and Convolutional Neural Network.” World Journal of Advanced Research and Reviews, vol. 21, no. 3, Mar. 2024, pp. 1009–19. https://doi.org/10.30574/wjarr.2024.21.3.0802.

GTZAN. “GTZAN Dataset - Music Genre Classification.” Kaggle, 24 Mar. 2020, www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification.

DistilBERT.

huggingface.co/docs/transformers/v4.47.0/en/model_doc/distilbert#overview.

Merritt, Rick. “What Is a Transformer Model? | NVIDIA Blogs.” *NVIDIA Blog*, 16 Oct. 2024, blogs.nvidia.com/blog/what-is-a-transformer-model.

Google Cloud Tech. “Transformers, Explained: Understand the Model Behind GPT, BERT, and T5.” *YouTube*, 18 Aug. 2021, www.youtube.com/watch?v=SZorAJ4I-sA.

Wikipedia contributors. “Transformer (Deep Learning Architecture).” *Wikipedia*, 6 Dec. 2024, en.wikipedia.org/wiki/Transformer_(deep_learning_architecture).

GTZAN Dataset - Music Genre Classification. *Kaggle*, 24 Mar. 2020, www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification.

MLMath.Io. “Math Behind SVM (Support Vector Machine) - MLMath.io - Medium.” *Medium*, 7 Dec. 2021, ankitnitjsr13.medium.com/math-behind-support-vector-machine-svm-5e7376d0ee4d.

9 Appendix

The following program was used for this investigation. Some insights to help write the code was taken from (*Andradaolteanu*) (*DistilBERT*)

Gtzan.py

```
import os

# Path to the Data folder
dataset_path =
'/Users/pranitha/.cache/kagglehub/datasets/andradaolteanu/gtzan-dataset-
music-genre-classification/versions/1/Data'

# Check the contents of the Data folder
if os.path.exists(dataset_path):
    data_contents = os.listdir(dataset_path)
    print("Contents of the Data folder:", data_contents)
```

```
else:
    print("Error: Data folder not found. Please verify the dataset.")

# Path to genres_original folder
dataset_path =
'/Users/pranitha/.cache/kagglehub/datasets/andradaolteanu/gtzan-dataset-
music-genre-classification/versions/1/Data/genres_original'

# Check if genres are present
if os.path.exists(dataset_path):
    genres = os.listdir(dataset_path)
    print("Genres found:", genres)
else:
    print("Error: genres_original folder not found.")
```

gcsvm.py

```
import os
import numpy as np
import librosa
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import time  # For tracking file processing time
from collections import Counter

# Path to genres_original folder
dataset_path =
'/Users/pranitha/.cache/kagglehub/datasets/andradaolteanu/gtzan-dataset-
music-genre-classification/versions/1/Data/genres_original'

# Check if genres are present
print("Checking dataset...")
if not os.path.exists(dataset_path):
    print("Error: genres_original folder not found. Exiting.")
    exit()

genres = os.listdir(dataset_path)
print("Genres found:", genres)


# Function to extract features from audio files
def extract_features(file_path):
    try:
        print(f"Extracting features from {file_path}...")
        y, sr = librosa.load(file_path, duration=30)  # Load 30 seconds of
audio
        y, _ = librosa.effects.trim(y)  # Trim silence

        # Extract features
        mfcc = np.mean(librosa.feature.mfcc(y=y, sr=sr, n_mfcc=20).T,
axis=0)
        chroma = np.mean(librosa.feature.chroma_stft(y=y, sr=sr).T, axis=0)
        mel = np.mean(librosa.feature.melspectrogram(y=y, sr=sr).T, axis=0)
        contrast = np.mean(librosa.feature.spectral_contrast(y=y, sr=sr).T,
axis=0)
        tonnetz =
```

```
np.mean(librosa.feature.tonnetz(y=librosa.effects.harmonic(y), sr=sr).T,
axis=0)

        return np.hstack((mfcc, chroma, mel, contrast, tonnetz))
    except Exception as e:
        print(f"Error during feature extraction for {file_path}: {e}")
        return None

# Extract features and labels
features, labels = [], []

print("Starting feature extraction...")
for genre in genres:
    genre_path = os.path.join(dataset_path, genre)
    print(f"Processing genre: {genre}...")
    for i, file in enumerate(os.listdir(genre_path)):
        if file.endswith('.wav'):
            file_path = os.path.join(genre_path, file)
            start_time = time.time()
            feature = extract_features(file_path)
            if feature is not None:
                features.append(feature)
                labels.append(genre)
            print(f"Processed file: {file} ({i +
1}/{len(os.listdir(genre_path))}) in {time.time() - start_time:.2f}
seconds")

print(f"Finished feature extraction for {len(features)} files.")

# Convert to NumPy arrays
print("Converting features and labels to NumPy arrays...")
X = np.array(features)
y = np.array(labels)
print(f"Features shape: {X.shape}, Labels shape: {y.shape}")

# Encode genre labels
print("Encoding genre labels...")
encoder = LabelEncoder()
y_encoded = encoder.fit_transform(y)
print("Encoded labels:", encoder.classes_)

# Scale the features
print("Scaling features...")
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data
print("Splitting data into training and testing sets...")
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_encoded,
test_size=0.3, random_state=42, stratify=y_encoded)
print(f"Training set size: {len(X_train)}, Test set size: {len(X_test)}")

# Train an SVM model
print("Training SVM model...")
svm = SVC(kernel='rbf', C=10, gamma=0.01)
svm.fit(X_train, y_train)

# Predict on the test set
print("Predicting on the test set...")
y_pred = svm.predict(X_test)
```

```
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"SVM Accuracy: {accuracy * 100:.2f}%")

# Confusion matrix
print("Generating confusion matrix...")
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=encoder.classes_, yticklabels=encoder.classes_)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Calculate genre-wise accuracy
correct_predictions = {genre: 0 for genre in encoder.classes_}
total_predictions = {genre: 0 for genre in encoder.classes_}

for true, pred in zip(y_test, y_pred):
    true_genre = encoder.inverse_transform([true])[0]
    pred_genre = encoder.inverse_transform([pred])[0]

    total_predictions[true_genre] += 1
    if true_genre == pred_genre:
        correct_predictions[true_genre] += 1


# Count the predicted genre distribution
svm_prediction_dist = Counter(y_pred)

# Plot pie chart for SVM predictions
plt.figure(figsize=(8, 8))
plt.pie(svm_prediction_dist.values(), labels=svm_prediction_dist.keys(),
autopct='%1.1f%%', startangle=90)
plt.title("Class-wise Prediction Distribution for SVM")
plt.axis('equal')  # Equal aspect ratio ensures that pie chart is circular.
plt.show()

# Alternatively, plot bar chart for SVM predictions
plt.figure(figsize=(10, 6))
plt.bar(svm_prediction_dist.keys(), svm_prediction_dist.values())
plt.title("Class-wise Prediction Distribution for SVM")
plt.xlabel("Genre")
plt.ylabel("Prediction Count")
plt.xticks(rotation=45)
plt.show()
```

gcdistilhuber.py

```
from transformers import pipeline
import librosa

# Load the pretrained pipeline
pipe = pipeline(
    "audio-classification",
    model="sanchit-gandhi/distilhubert-finetuned-gtzan",
    device=-1  # Force CPU usage
)
```

```
# Preprocess the audio file with Librosa
def preprocess_audio(file_path, target_sr=16000):
    """
    Load an audio file, resample it to the target sample rate, and return
the waveform as a numpy array.
    """
    y, sr = librosa.load(file_path, sr=target_sr)  # Load and resample
audio to 16kHz
    return y

# Path to your audio file
file_path =
"/Users/pranitha/.cache/kagglehub/datasets/andradaolteanu/gtzan-dataset-
music-genre-
classification/versions/1/Data/genres_original/pop/pop.00001.wav"

# Preprocess the audio
audio_waveform = preprocess_audio(file_path)

# Pass the preprocessed waveform directly to the pipeline
result = pipe(audio_waveform, sampling_rate=16000)
print(result)
```

gcDHubert.py

```
import os
import numpy as np
from transformers import pipeline
import librosa
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import time  # For tracking time
from collections import Counter

# Define dataset path
dataset_path =
"/Users/pranitha/.cache/kagglehub/datasets/andradaolteanu/gtzan-dataset-
music-genre-classification/versions/1/Data/genres_original"

# Initialize the DistilHuBERT pipeline
pipe = pipeline(
    "audio-classification",
    model="sanchit-gandhi/distilhubert-finetuned-gtzan",
    device=-1  # Force CPU usage
)

# Function to preprocess audio with librosa
def preprocess_audio(file_path, target_sr=16000):
    """
    Load an audio file, resample it to the target sample rate, and return
the waveform as a numpy array.
    """
    y, sr = librosa.load(file_path, sr=target_sr)  # Load and resample
audio to 16kHz
    return y

# Lists to store true and predicted labels
true_labels = []
```

```
predicted_labels = []

# Get all genres
genres = os.listdir(dataset_path)

print("Starting evaluation on GTZAN dataset...")
start_time = time.time()

# Loop through all genres and files
for genre_idx, genre in enumerate(genres, start=1):
    genre_path = os.path.join(dataset_path, genre)
    print(f"Processing genre {genre_idx}/{len(genres)}: {genre}...")
    genre_start_time = time.time()

    for file_idx, file in enumerate(os.listdir(genre_path), start=1):
        if file.endswith(".wav"):
            file_path = os.path.join(genre_path, file)
            try:
                # Preprocess the audio
                print(f"  [{genre}] Preprocessing file {file_idx}: {file}")
                audio_waveform = preprocess_audio(file_path)

                # Predict using the pipeline
                print(f"  [{genre}] Predicting genre for {file}...")
                result = pipe(audio_waveform, sampling_rate=16000, top_k=1)

                # Store the top prediction and true label
                predicted_labels.append(result[0]["label"])
                true_labels.append(genre)

                print(f"  [{genre}] Prediction for {file}:
{result[0]['label']} (score: {result[0]['score']:.4f})")

            except Exception as e:
                print(f"  Error processing {file_path}: {e}")

    print(f"Finished processing genre {genre} in {time.time() -
genre_start_time:.2f} seconds.\n")

print(f"Finished evaluation on GTZAN dataset in {time.time() -
start_time:.2f} seconds.")

# Calculate accuracy
accuracy = accuracy_score(true_labels, predicted_labels)
print(f"DistilHuBERT Accuracy on GTZAN dataset: {accuracy * 100:.2f}%")

# Generate confusion matrix
print("Generating confusion matrix...")
conf_matrix = confusion_matrix(true_labels, predicted_labels,
labels=genres)

# Plot confusion matrix
plt.figure(figsize=(12, 10))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=genres, yticklabels=genres)
plt.title("Confusion Matrix: DistilHuBERT")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# Calculate genre-wise accuracy
```

```
correct_predictions = {genre: 0 for genre in genres}
total_predictions = {genre: 0 for genre in genres}

for true, pred in zip(true_labels, predicted_labels):
    total_predictions[true] += 1
    if true == pred:
        correct_predictions[true] += 1


# Count the predicted genre distribution for DistilHuBERT
distilhubert_prediction_dist = Counter(predicted_labels)

# Plot pie chart for DistilHuBERT predictions
plt.figure(figsize=(8, 8))
plt.pie(distilhubert_prediction_dist.values(),
labels=distilhubert_prediction_dist.keys(), autopct='%1.1f%%',
startangle=90)
plt.title("Class-wise Prediction Distribution for DistilHuBERT")
plt.axis('equal')  # Equal aspect ratio ensures that pie chart is circular.
plt.show()

# Alternatively, plot bar chart for DistilHuBERT predictions
plt.figure(figsize=(10, 6))
plt.bar(distilhubert_prediction_dist.keys(),
distilhubert_prediction_dist.values())
plt.title("Class-wise Prediction Distribution for DistilHuBERT")
plt.xlabel("Genre")
plt.ylabel("Prediction Count")
plt.xticks(rotation=45)
plt.show()
```